A Trustworthy Mobile Agent Infrastructure for Network Management

Alexandros Koliousis
Department of Computing Science
University of Glasgow
17 Lilybank Gardens, Glasgow G12 8RZ
Email: koliousa@dcs.gla.ac.uk

Joseph Sventek
Department of Computing Science
University of Glasgow
17 Lilybank Gardens, Glasgow G12 8RZ
Email: joe@dcs.gla.ac.uk

***Abstract*—Despite several advantages inherent in mobile-agent-based approaches to network management as compared to traditional SNMP-based approaches, industry is reluctant to adopt the mobile agent paradigm as a replacement for the existing manager-agent model; the management community requires an evolutionary, rather than a revolutionary, use of mobile agents. Furthermore, security for distributed management is a major concern; agent-based management systems inherit the security risks of mobile agents. We have developed a Java-based mobile agent infrastructure for network management that enables the safe integration of mobile agents with the SNMP protocol. The security of the system has been evaluated under agent to agent-platform and agent to agent attacks and has proved trustworthy in the performance of network management tasks.**

I. Introduction

Mobile agents, software processes that can migrate between hosts for execution, have been extensively suggested in the literature as a solution enabling decentralized, flexible, and scalable distributed applications [1]. The mobile agent paradigm enables the migration of code, state, and data of an agent from one host to another for execution. The migration process of an agent is constrained by its itinerary path. It is extremely difficult to evaluate the trustworthiness of agent code execution [2]. Thus, additional mechanisms are required to ensure trust in an agent-based system.

Network management systems are an example of large, distributed applications. Currently, most management systems for IP networks are centralized, based on the client-server paradigm. The Simple Network Management Protocol (SNMP) is the dominant protocol for the interactions between the client, the management station, and the servers, SNMP agents that reside on the network elements; each SNMP agent serves as a proxy between the manager and the Management Information Base (MIB) of the device. The third version of the protocol attempts to address the security weakness of its predecessors. Nonetheless, the efficacy of SNMP is contrasted with the requirements of modern management environments [3]. In order to overcome issues of response implosion, scalability, flexibility, heterogeneity, and robustness in these centralized management systems, the network management community has primarily focused on distributed management architectures[4].

Although code mobility has always been a candidate to address such requirements [5], the industry is still reluctant to adopt the mobile agent paradigm as a replacement to the existing client-server model, when applicable. Independent of any market considerations, it must be possible to establish the trustworthiness of mobile code in order to assure the correct workings of the system. Otherwise, the manager essentially delegates management responsibility to a virus. Furthermore, the sheer number of SNMP devices mandates that new distributed services must comply with the existing structure. Since most network managers think about their management tasks in terms of SNMP MIBs, a trustworthy mobile agent system should concentrate upon enabling mobile access to such data as an evolutionary step [6]; this is to be contrasted with the paradigm shift that is demanded by most mobile-agent-based management systems[1]. To address these requirements we have developed a Java-based mobile agent architecture that safely integrates mobile agents and SNMP agents. The Ajanta mobile agent framework [8] has been used as the underlying agent-platform; the VMC concept [6] has been used for the requisite integration with SNMP agents.

We investigate the behavior of malicious mobile code in network management scenarios to gain an overall understanding of the security issues raised by the use of mobile agents in distributed applications. We show that the resulting network management system can assure the trustworthiness of mobile agents while it can provide the necessary infrastructure to support network management functions, as defined in the OSI management model. The proposed scheme complements existing work on agent-based management systems and is applicable to other mobile code paradigms, viz. Remote Evaluation (REV), Code on Demand (CoD), and constrained mobility [9]. The existing technologies provide an acceptable level of trust, and thus, the proposed system addresses the two issues described above that prevent the adoption of mobile agents for network management systems.

II. Security in agent-based management systems

Network management systems provide security mechanisms to enforce the safety requirements of an application. In order

[1]For example, the agent-based management system in [7] requires that each node visited by a mobile agent must present management data and services using tuple spaces.

1-4244-0799-0/07/$25.00 ©2007 IEEE

to achieve this functionality, the system **must** ensure that management is performed in a secure way. It must guarantee the identity, authority, confidentiality, and integrity of management information traversing the network to an acceptable level of trust.

An agent-based management system conceals information within an agent. A management agent α_i can be described by the quadruple $\alpha_i = \{code_i, data_i, itinerary_i, credentials_i\}$, where $credentials_i$ is the set of credentials for the authentication and authorization of the agent. The credentials contain information about the agent's name, its creator, and its owner. Upon successful authentication and authorization, agent code, including its state, is executed within an agent-platform, which provides an interface to an SNMP agent. Mobile code for network management has introduced new security threats that fall into four categories: agent to agent-platform attacks, agent to agent attacks, agent-platform to agent attacks, and external to agent-platform attacks.

Authentication and authorization mechanisms based on cryptographic keys are the two basic components of a secure distributed application. They can be used to prevent disclosure, modification, or unauthorized access to sensitive management information by malicious users; they can also prevent repudiation threats. However, these mechanisms are inadequate to ensure the trustworthiness of agent-based management systems; migrated agent code may not reflect the initial intentions of the owner. Furthermore, there is no reliable way to ensure that the agent-platform will run the agent to completion. The system is also susceptible to Denial of Service (DoS) attacks.

An agent to agent-platform attack can be launched against the CPU and memory resources of the hosting system. For example, an agent consuming CPU cycles will result in resource starvation; the system becomes livelocked because other agents are unable to run to completion. Besides physical resources, attacks could target application-specific resources, i.e. the SNMP agent, the execution environment itself, e.g. the Java Virtual Machine (JVM), or other management agents. The system can become unpredictable, or even terminate, because of the malicious behavior of the agent. Finally, a malicious agent can create multiple copies of itself, its migratory nature allowing the agent to behave as a "worm" program. A compromised platform is a perfect vehicle for a malicious agent to infect the entire network, even in the case of constrained mobility.

The agent execution environment must prevent such abuse of system resources. Resource restrictions can be enforced upon mobile agents either globally, at agent creation, or locally, at execution by the receiving agent-platform [10]. Global restrictions are included in the agent's credentials and propagated to the agent-platform. They can be expressed in the form of a contract, or a formally verified proof [11]; in the latter case, a proof of compliance with the rules, as imposed by the hosting platform, is generated off-line. Although off-line verification is theoretically sound, it lacks completeness; resource restrictions are usually estimates of the actual resource consumption. Furthermore, it is impossible to determine if the program will terminate, due to the unsolvability of the Halting Problem. An alternative to off-line verification is on-line monitoring of an agent during its execution phase.

Agent attacks can be oriented towards other agents of the system. A malicious agent may disrupt the normal execution of another agent by repeatedly sending it messages. Furthermore, a malicious agent can gain unauthorized access to sensitive data carried by other agents. This may result in unauthorized disclosure of information, but it can also affect the reliability of the system by altering sensitive information. The inter-agent communication model between two management agents implies that one agent plays the role of the server, providing management information, and thus must be protected by similar mechanisms as agent-platforms.

In a reverse of roles, mobile agents are susceptible to attacks launched by malicious agent-platforms. Peer authentication mechanisms can verify the identity of the agent-platform; however, additional security mechanisms are required to ensure agent safety in a foreign execution environment. Upon acceptance, an agent exposes its code, state, and data to the agent-platform, and thus any information about the network and its vulnerabilities. A malicious platform can ignore requests, introduce new tasks or unacceptable delays for tasks, or even deny agent execution. It can falsify management information or even tamper with the agent's code. Cryptographic mechanisms exist [2] to detect tampering of mobile agents by malicious platforms, *a posteriori*.

The mobile agent paradigm entails the migration of code, state, and data of an agent to another hosting platform for execution. The attacks discussed so far are mounted after the completion of the migration process. However, agents, agent-platforms, and the hosting network device can be accessed remotely; external to agent-platform attacks refer to attacks launched by entities external to the mobile agent system. An agent system is an exemplar case of a distributed application and thus it must be protected against remote attacks, such as eavesdropping, unauthorized access, and denial of network resources. By design, management platforms must be protected from malicious network users.

III. Related work

The SNMPv3 set of RFCs provides a framework for incorporating security enhancements into the existing SNMP functionality [12]. It defines two models, the user security model and the view-based access control model, for the authentication and authorization of principals in a distributed management system. The SNMP security model can be complementary to mobile code security. However, it cannot satisfy the safety requirements of an agent-based management application. Additional mechanisms are required to prevent code mobility threats. For example, the IETF Script MIB, an SNMP extension to enable constrained mobility in SNMP-based network management systems, has been extended [13] to enforce resource restrictions to management scripts.

A hybrid management system, integrating Aglets with the SNMP protocol, has been proposed in [14]. The Aglets Work-

bench security [15] is based on an authorization language. Besides defining identification information, such as the name, owner, or creator of an agent, the Aglets model can assign roles and role hierarchies to agents for access control. Estimated resource limitations can be enforced at creation time as global restrictions, and are included in the credentials of the agent. The security model has been extended to include cryptographic functions to protect agents from malicious platforms. The system is still susceptible to certain agent to agent-platform attacks. Furthermore, by allowing aglets to directly issue SNMP commands, the system cannot guarantee the safe integration of mobile agents with the SNMP protocol.

A secure agent-based network management system has been proposed in the TINMAN architecture [16]. Resource safety is enforced by introducing resource utilization certificates. A prediction of the resource utilization of an agent is calculated off-line, capturing the resource usage behavior of the agent. Upon migration, the hosting platform verifies if the certificate complies with the local security policies. Assertions that fail off-line verification are monitored on-line by raising run-time exceptions. Currently, only memory and timing constraints are imposed by the system. Agents in the TINMAN architecture directly access the MIB data of the device; the underlying resource is protected only by the SNMP security model, and thus, it is susceptible to DoS attacks.

Resource utilization is monitored on-line in the Secure and Open Mobile Agents (SOMA) programming framework [17]. The SOMA architecture introduces a monitoring component that maintains information about the resource utilization of the operating system, e.g. CPU and memory usage, disk space, number of open files, number of TCP connections, total number of UDP/IP packets etc. Information about the resources of the host are collected using the Java Native Interface (JNI), the Java Virtual Machine Profiler Interface (JVMPI), or the SNMP agent of the running host.

Although our JVMPI-based memory monitoring mechanism has some resemblance to the SOMA monitoring component, there are key differences with our management paradigm. First, integration of SOMA mobile agents with system resources, e.g. the SNMP agent, is based on weak mobility, via CORBA objects. Secondly, our monitoring mechanism is based strictly on the JVM profiler; by design, our security model is contained within the agent-platform environment rather than relying on external resources.

IV. The System Architecture

We have developed a Java-based mobile agent architecture for SNMP-based network management. The architecture consists of three components, a mobile agent system, an SNMP agent, and an interface for the safe integration of mobile agents with SNMP. Figure 1 illustrates the system architecture.

Java has been widely accepted as a language for programming mobile agents, primarily because it can address issues of interoperability and heterogeneity of network devices. It provides a serialization mechanism to support agent migration. An agent class must implement the `Serializable` interface.

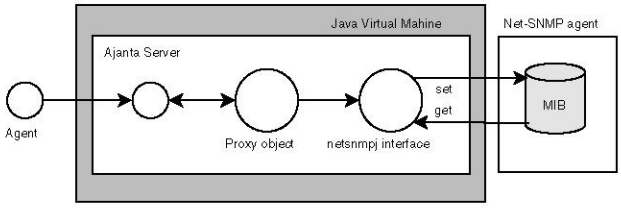

Fig. 1. The system architecture.

An agent object is represented as a series of bytes; upon acceptance, a new object is created with the same state as the original agent object. Java security is based on the sandbox execution model. The Java security model [18] is customizable and enables the creation of security policies and protection domains for agents according to the requirements of the application. It consists of three components, a class loader, a bytecode verifier, and an access controller.

The class loader mechanism creates different name spaces for each class that is executed. Protection is enforced by checking if the code has appropriate privileges to access a particular class that resides locally, or on a remote code base. The bytecode verifier ensures that a class is safely typed. For example, it proves that no illegal casting occurs, or that final classes are not overridden. The bytecode verifier operates on data types, not on data values; the verification process cannot prevent misuse or abuse of the system resources by type-safe malicious code. Upon receipt of the code for an agent, the access controller creates a protection domain for that code. A protection domain groups all permissions that apply to the owner of the code.

The Ajanta mobile agent system [8] has been chosen to serve as the core of our system architecture because it provides an overall security model, based on Javas sandbox execution model, to address the security issues raised by agent to agent-platform and agent to agent attacks. The Ajanta security model provides three cryptographic mechanisms based on selective encryption of agent components, append-only data, and selective state exposure to detect tampering of an agents code or data by malicious platforms; unfortunately, these are insufficient to fully address agent-platform to agent attacks. This work focuses on prevention, as this is the most important issue preventing the adoption of MA technology in network management applications. Therefore, the proposed management framework has been developed under the assumption that agent-platforms are trusted entities of the system. As such, agent-platform to agent attacks are beyond the scope of this paper; we revisit this assumption in Section VI. Note that this assumption enables us to include our on-line resource monitoring mechanism in the architecture and conceal the agent-based management system from malicious external entities.

A. Security in the Ajanta system

The Ajanta system [8] is a Java-based framework for building mobile agent applications. Ajanta provides a reliable transfer protocol for agent migration. Agent migration is

based on a Publish-Discovery-Bind model. Agent-platforms advertise (publish) themselves to a Name Registry service. Given the name of the destination agent-platform, the Name Registry returns its physical location (the discovery process). The Agent Transfer Protocol is invoked between the current and the destination platform (the bind process) when an agent requests to migrate to the destination host. The agent transfer can be authenticated or encrypted, optionally, to protect the agent while in transit over an insecure channel.

Ajanta implements a Public Key Infrastructure (PKI) for agent authentication. The Name Registry server acts as the public key distribution service. The authentication procedure can be one-way or mutual. It is based on a challenge-response mechanism using digitally signed nonces to prevent replay attacks and support single sign-on authentication along the agent's itinerary path. Upon request for migration, the current platform is challenged by the destination platform with a nonce to prove its claimed identity. The nonce is digitally signed and returned to the destination platform for verification. The two entities can remain in agreement for the next signed challenge value to maintain authentication in the session.

Protection of agent-platforms from malicious agents is provided by using a low-level agent isolation model based on Java protection domains. Each agent is constrained in a unique thread group. An agent can access or modify threads only within its thread group. Thus, an agent cannot directly manipulate other agent threads on the machine. A separate class loader is assigned to each agent. By assigning a separate class loader instance to each agent, the system prevents a malicious agent from altering object classes of co-located agents. Furthermore, upon request for a class, the class loader first checks the local classpath, before downloading it from a remote code base, enforcing this way the integrity of the underlying Java Virtual Machine.

Ajanta implements a proxy-based mechanism to protect application-specific resources of the system. Instead of providing direct access to resources, Ajanta creates a proxy object with a reference to the actual resource implementation. Resources are advertised in a Resource Registry. Given the name of a resource, a distinct proxy object is created for every agent that requires access to that resource. Operation visibility in the proxy is dependant upon the credentials of the agent; methods are enabled or disabled based on an simple Access Control List (ACL) lookup. The proxy object is declared as `private` and `transient`, to prevent illegal access to methods and serialization of resource object respectively. Furthermore, the proxy class cannot be cloned by a malicious agent.

B. Integration of mobile agents with the SNMP protocol

The Virtual Managed Component [6] (VMC) is an interface that enables communication between mobile agents and SNMP agents. An agent has the ability to access the management information of the device through that interface. The notion of VMC has been previously proposed for the integration of mobile agents with the SNMP protocol [5]. This enables the development of lightweight mobile agents without carrying any SNMP capabilities, such as BER encoding and decoding, but only higher level tasks. One approach is to have mobile agents directly issue requests to the SNMP agent [14], [16]. However, enabling direct communication with the SNMP agent implies a security risk. The latter would be protected only by the authentication and authorization mechanisms of the SNMPv3 security model.

We have developed a Java class that implements methods for reading (the `get` command) or modifying (the `set` command) objects from the MIB of the device. Access to the class is granted via Ajanta's proxy mechanism. Two types of agents have been defined in our system architecture, *read-only* and *read-write* agents. Read-only agents are mobile agents that can only monitor management objects; read-write agents are able to alter MIB data. Upon successful authentication and authorization, the Ajanta system returns a reference to the proxy class with the `get` and `set` methods enabled or disabled based on the agent's credentials. We have extended Ajanta's authorization mechanism by introducing role hierarchies, based on agent owners, creation platforms, and agent names, in an attempt to enable delegation of authority via hierarchical access control lists. For example, an agent can access a proxy method only if its creator has such privilege. An additional level of authorization can also be enforced by the SNMP agent itself, based on the SNMPv3 user security and view-based access control model; if activated, the agent credentials must be enriched with the security information required by the SNMP agent and passed as parameters during a method invocation.

V. Evaluating the efficacy of the scheme

The efficacy of the system has been evaluated under agent to agent-platform and agent to agent attacks. In particular, we study the behavior of the system after a malicious agent has passed the authentication and authorization mechanisms, in an attempt to evaluate the trustworthiness of mobile code execution. An agent that has gained unauthorized access to the system can launch attacks against the system resources, or alter sensitive information of the management system. Our testbed environment consists of five Linux machines, each running an Ajanta server. Attacks are mounted by one agent-platform hosting one or more malicious agents. Network devices can be remotely accessed by other agent platforms only via migrated code; external entities are concealed from our system implementation.

A. A CPU-bound agent

A malicious agent can compromise an agent-platform by consuming large amounts of CPU cycles. A malicious agent running a CPU-bound operation at the highest priority prevents other mobile agents from running to completion. The agent-platform becomes livelocked.

A CPU DoS attack has been launched in our testbed environment. In Ajanta, there is an 1-to-1 correspondence between agents and Java threads. Threads are executed by the JVM based on a fixed-priority scheduling algorithm. The

Ajanta system does not enforce any restrictions on thread priorities. A malicious agent increases its priority level and begins a CPU-intensive computation in an infinite loop. A CPU-friendly agent is accepted by the system. It is expected that the latter agent will be starved of CPU resources. Java enforces priority restrictions via the `modifyThread` runtime permission. However, even if agents are restricted to run with default priorities, the malicious agent can still run infinitely, when all other agents have finished execution. A policy is required to enforce an upper-bound restriction on CPU cycles.

A control mechanism has been developed that enforces limitations on agent execution time. The mechanism is implemented as a periodic task, running at regular intervals. If an agent's execution time is found to exceed a predefined threshold $T_{threshold}$, the agent server terminates the agent. We compute $T_{threshold}$ as follows: if T_{avg} is the average execution time of a mobile agent, and N the number of currently active agents on the agent-platform, then an agent should run to completion by $T_{threshold} = (N+1)T_{avg}$. The threshold is a dynamic variable updated each time an agent execution begins or, respectively, ends.

The execution time of an agent is proportional to the number of agents currently hosted on the device. In our system implementation, there is an 1-to-1 correspondence between a Java thread and a Linux thread. Linux implements a time-sharing scheduling algorithm; Java priorities are ignored. Each agent thread is assigned an equal time quantum for the CPU; once the quantum expires, the next ready-to-run thread is executed. However, even if the system implemented a different scheduler, our mechanism would again prevent malicious agents from consuming all CPU cycles. It must be noted that the CPU-friendly agent manages to run to completion, without activating our control mechanism. The attack can also be prevented by the underlying operating system, at the expense that agent priorities are ignored. However, this particular mapping may be inappropriate, or infeasible in another system implementation. Also note that if one relies upon the operating system in this way, the malicious agent would continue to consume CPU cycles after all CPU-friendly agents have terminated.

B. A memory-bound agent

A DoS attack can be launched against the memory resources of the receiving agent-platform. A malicious agent can consume the available memory of the system by aggressively allocating objects in heap memory. The agent-platform will then be left with insufficient resources to serve subsequent memory requests from friendly agents.

We have developed a memory management mechanism to enforce restrictions on memory consumption. Control policies are primarily based on agent memory allocation rates. An agent allocation rate is defined as the increase of an agent's memory usage during the time between two successive allocation requests. Memory requests are aperiodic events, and thus we observe large fluctuations in the allocation rate. We use a weighted, exponentially smoothed average allocation rate to reduce these fluctuations and avoid penalizing non-malicious agents that simply want to instantiate an array for storing and processing MIB variables. Indeed, the moving average only punishes "aggressive" agents (Figure 2).

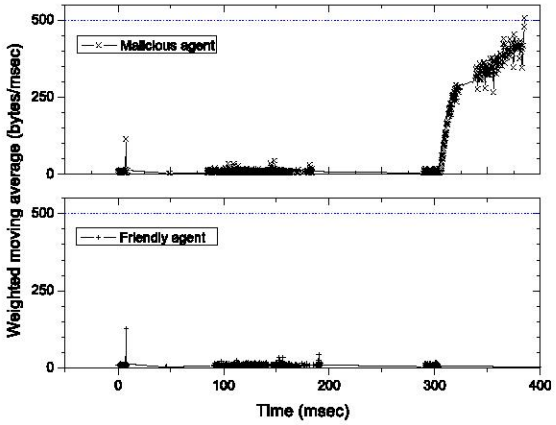

Fig. 2. A malicious agent is terminated once its average allocation rate exceeds a predefined threshold. The memory-friendly agent, although it consumes the same amount of memory as the malicious one, is not penalized by the memory management system.

Two additional policies are enforced, a memory usage upper-bound restriction for agents, and a maximum object size restriction for a single allocation request. Mobile agents that violate any of the three policies are considered malicious and are terminated by the agent-platform. There is no guarantee that the system will preserve sufficient memory for a mobile agent to run to completion. However, the system reserves sufficient memory to instantiate new agents on the agent-platform. Memory exceptions can then be handled by the agent itself.

Our memory management mechanism, similar to the CPU monitoring mechanism, exploits on-line monitoring to control resources in the management system; they enforce the trustworthiness on mobile code execution, rather than abide by the resource usage specification provided by the the agent's creator. The latter, although usually an estimate of the required resources, can be complementary to our approach.

C. Cloning an agent

Mobile agents may exhibit similar behavior to "worm" programs. A malicious agent can create multiple copies of itself and saturate the resources of the hosting system, or launch a replay attack by self-propagating to other hosts of the network management system. Agent cloning functionality is inherent in mobile agent systems that support the Master-Slave migration pattern. We define legal clones as children, derived from agents implementing the Master-Slave migration pattern. We wish to restrict malicious agents from creating illegal clones in the system.

A malicious agent can generate a replica by invoking the `Object.clone()` method, inherited by default by every class

of the system. Taking advantage of Java's serialization mechanism, upon which agent migration is based in Ajanta, the malicious agent can create a deep copy of itself. A deep copy, in contrast to a shallow copy, contains a copy of all the objects referenced by the agent. The `notClonableAgent` class has been introduced to the system to prevent malicious agents from cloning. The class overrides and, eventually, finalizes the `clone()` method. Agents derived from this class cannot create replicas of themselves. The mechanism is also sufficient to prevent malicious agents from cloning other agents that reside on the same platform.

Whilst our mechanism prevents illegal replicas, legal clones still pose a security threat. Yet, the system is sufficiently secure to prevent malicious actions by legal clones. Ajanta assigns a unique name, in ascending numerical order, to each newly created child in order to distinguish it from its parent. The system can impose a restriction on the number of clones an agent can create. Furthermore, the security mechanisms of the system can prevent the saturation of the agent-platform by imposing restrictions on the resource usage. Finally, replay attacks can be successfully prevented by the authentication mechanism of the Ajanta system; the use of randomly generated nonces in the challenge-response exchange guarantees the uniqueness of the migration process. Replay attacks can also be prevented by timing constraints [14]. However, we do not wish to impose time restrictions on our application.

D. Protection of the management application resources

An agent-based management system must ensure the safe integration of mobile agents with the SNMP interface to the underlying management agent. A first level of security in our system is enforced by the Java language semantics. Ajanta's proxy mechanism utilizes the Java security model to prevent misuse of the underlying resources, in this case the SNMP interface. By declaring the reference in the resource class as `private`, a malicious agent cannot access the interface methods directly. Also, the proxy class is directly derived from the `Object` class; any attempt to typecast (other than downcasting) the resource object will raise an exception. The reference to the SNMP interface is also declared `transient` to prevent malicious agents from serializing, cloning, and finally creating an illegal copy, exposing the resource to an intruder in this way. Furthermore, the class loading mechanism prevents malicious agents from installing their own resource proxy class. Finally, the access controller prevents agents from creating and installing their own class loader.

We consider two additional attacks in our evaluation. First, a malicious agent can launch a DoS attack against the SNMP agent by repeatedly requesting MIB variables, in an attempt to saturate the resource. Second, a malicious agent can completely bypass the proxy mechanism and access the SNMP agent directly. In the first scenario, the malicious agent is restricted by the CPU monitoring mechanism of the system and is eventually preempted. Furthermore, resource-friendly agents are able to perform network management tasks, even if a malicious one tries to monopolize the resource. The Ajanta system can impose a time restriction on the lifetime of the proxy object, as an additional security mechanism. In the second scenario, we have launched a malicious agent with embedded SNMP functionality. By enforcing the rule that mobile agents are not allowed to directly open sessions with the SNMP agent such attacks fail because agents that attempt to bypass the proxy-based mechanism simply cannot communicate with the SNMP agent.

An agent to agent-platform attack can target system-wide resources, such as user files, system logs, or even the agent execution environment, i.e. the Java Virtual Machine (JVM). We believe that a mobile management agent does not require access to a particular system file to perform a SNMP network management task. Nevertheless, if such access is mandated, it should be granted via a resource interface, similar to the VMC concept for accessing SNMP agents. Agents must be isolated and constrained to perform only their designated tasks. As a representative example, in our attack scenario a malicious agent attempts to replace a class of the current Java installation with a malicious one. By tampering with the bytecode of the available classes, a malicious agent can compromise the receiving host. Illegal access is prevented by Ajanta's proxy mechanism, Java security policies, and the underlying operating system.

E. Agent to agent attacks

A malicious mobile agent can launch attacks against other agents of the system. An agent to agent attack can target agents co-located in the agent-platform, or other remote agents. In either case, a malicious agent can disrupt, or even terminate the execution of an agent.

A malicious agent may attempt to disrupt the execution of other co-located agents by manipulating the currently running threads of the agent-platform. In our attack scenario, the malicious agent was instructed to terminate every agent co-located on the agent-platform. The Ajanta system assigns a unique thread group to each mobile agent. Agent threads are protected by Ajanta's security manager and class loading mechanisms. Given a thread group, the security manager checks if the calling program has permissions to manipulate the thread group. Access control is invoked when a program requests permission to create, interrupt, or terminate a thread group; otherwise, a security exception is raised.

In the mobile agent paradigm, the role of the client and server is interchangeable between two communicating agents. For example, in a network management scenario a stationary agent can serve as an access point for a management agent to monitor or control a network element. Inter-agent communication in Ajanta is based on the proxy mechanism. An agent presents an interface to the system containing the methods that other agents are allowed to invoke. Upon request, the system returns a proxy object containing a reference to the actual agent object. The same security policies that have been enforced to protect the management application resources, i.e. the SNMP interface, can also used to prevent a malicious agent

TABLE I
AGENT TO AGENT-PLATFORM RESULTS

TYPE OF ATTACK	PREVENTION MECHANISM
CPU Denial of Service	CPU control mechanism
Memory Denial of Service	Memory control mechanism
Agent cloning	The `notClonableAgent` class
Unauthorized SNMP method invocation	Resource is referenced as `private`
Typecasting the SNMP resource	Resource is derived from the `Object` class
Cloning the SNMP resource	Resource is referenced as `transient`
A Denial of Service against the SNMP agent	CPU control mechanism
Bypass the VMC interface	Restricted communication with the SNMP agent
Manipulate the execution environment (e.g. JVM)	proxy mechanism; Java security; OS security

TABLE II
AGENT TO AGENT RESULTS

TYPE OF ATTACK	PREVENTION MECHANISM
Terminate (or manipulate) co-located agents	Java protection domains; class loader mechanism
Unauthorized method invocation of remote agents	Proxy mechanism; access control
Terminate (or manipulate) remote agents	Authentication and authorization control

from accessing sensitive information, tampering with the code, or launching a DoS attack against the remote agent.

F. Summary

Table I and Table II summarize our results from evaluating the trustworthiness of the proposed scheme under agent to agent-platform and agent to agent attacks, respectively. In particular, the system prevents Denial of Service attacks against the CPU, memory, and SNMP resource of the system by enforcing upper-bound restrictions on their utilization. It utilizes the Java security model to prevent typecasting, cloning, or overriding the VMC interface and directly access the underlying SNMP agent. Furthermore, an agent cannot attack its execution environment, i.e. the Java Virtual Machine, or the underlying operating system. Finally, malicious agent attacks on other co-located or remote agents of the system are also prevented.

VI. CONCLUSIONS AND FUTURE WORK

Mobile agents have introduced a powerful software paradigm, able to perform complex management tasks while addressing issues of scalability and flexibility in distributed network management system. However, proposed solutions often disregard the wide acceptance of legacy protocols; there has been a substantial investment in SNMP-based systems and thus, transition to agent-based management should be evolutionary, rather than revolutionary.

Mobile agent systems have introduced several security threats that must be taken into account when designing an agent-based distributed system. We have enumerated the possible security attacks and proposed a number of mechanisms for their prevention, in the context of a network and system management application. Our work draws from previous work in two different disciplines, mobile agent security and mobile agents for network management. The security mechanisms discussed in this paper can be generalized and applied to other agent-based distributed applications. For example, the proxy mechanism in Ajanta has been designed to protect any type of resource and was not designated for SNMP-based management systems. We have applied these mechanisms to network management systems and emphasize the importance of security for agent-based network management.

The discussion in Section V has shown that the implemented management infrastructure prevents agent to agent-platform and agent to agent attacks (see Tables I and II). This has been experimentally demonstrated by securely integrating a mobile agent platform with an SNMP agent; we can guarantee the trustworthiness of the mobile agent infrastructure as well as provide an evolutionary path for network management systems to begin to take advantage of the benefits of mobile agent technology.

As indicated in Section IV, this work assumed that agent-platform to agent attacks could be ignored. While there has been some research into detection of tampering *a posteriori*, there is a need to investigate methods by which an agent can

determine during execution that it is experiencing malicious modification of its state [19].

REFERENCES

[1] A. Fuggetta, G. P. Picco, and G. Vigna, "Understanding code mobility," *IEEE Transactions on Software Engineering*, vol. 24, no. 5, pp. 342–361, 1998.

[2] W. Jansen and T. Karygiannis, "Mobile agent security," National Institute of Standards and Technology, Computer Security Division, Gaithersburg, MD 20899, NIST Special Publication 800-19, 1999.

[3] J. Schonwalder, A. Pras, and J.-P. Martin-Flatin, "On the future of internet management technologies," *Communications Magazine, IEEE*, vol. 41, no. 10, pp. 90–97, October 2003.

[4] G. Goldszmidt and Y. Yemini, "Distributed management by delegation," in *ICDCS '95: Proceedings of the 15th International Conference on Distributed Computing Systems*. Washington, DC, USA: IEEE Computer Society, 1995, p. 333.

[5] A. Bieszczad, T. White, and B. Pagurek, "Mobile agents for network management," *IEEE Communications Surveys*, vol. 1, no. 1, 1998.

[6] B. Pagurek, Y. Wang, and T. White, "Integration of mobile agents with SNMP: Why and how," in *NOMS 2000: IEEE/IFIP Network Operations and Management Symposium*, Honolulu, Hawaii, April 2000, pp. 609–622.

[7] G. Cabri, L. Leonardi, and F. Zambonelli, "Mobile agent coordination for distributed network management," *Journal of Network and Systems Management*, vol. 9, no. 4, pp. 435–456, 2001.

[8] N. M. Karnik and A. R. Tripathi, "Security in the ajanta mobile agent system," *Software – Practice and Experience*, vol. 31, no. 4, pp. 301–329, 2001.

[9] C. Bahoris, A. Liotta, and G. Pavlou, "Evaluation of constrained mobility for programmability in network management," in *DSOM '00: Proceedings of the 11th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*. London, UK: Springer-Verlag, 2000, pp. 243–257.

[10] J. Tardo and L. Valente, "Mobile agent security and Telescript," in *IEEE Compcon'96. 'Technologies for the Information Superhighway' Digest of Papers*, February 1996, pp. 58–63.

[11] G. C. Necula, "Proof-carrying code," in *POPL'97: 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Langauges*, Paris, January 1997, pp. 106–119.

[12] D. Harrington, R. Presuhn, and B. Wijnen, "An architecture for describing simple network management protocol (SNMP) management frameworks," Network Working Group, Request for Comments, RFC 3411, 2002.

[13] A. da Rocha, C. A. da Rocha, and J. N. de Souza, "Script MIB extension for resource limitation in SNMP distributed management environments," in *Telecommunications and Networking - ICT 2004, 11th International Conference on Telecommunications*, Fortaleza, Brazil, 2004, pp. 835–840.

[14] A. Pashalidis and M. Fleury, "Secure network management within an open-source mobile agent framework," *Journal of Network and Systems Management*, vol. 12, no. 1, pp. 9–31, March 2004.

[15] G. Karjoth, D. B. Lange, and M. Oshima, "A security model for aglets," *IEEE Internet Computing*, vol. 1, no. 4, pp. 68–77, 1997.

[16] W. Yu and A. K. Mok, "Enforcing resource bound safety for mobile SNMP agents," in *ACSAC'02: 18th Annual Computer Security Applications Conference*, Las Vegas, Nevada, December 2002, pp. 69–77.

[17] P. Bellavista, A. Corradi, and C. Stefanelli, "How to monitor and control resource usage in mobile agent systems," in *3rd IEEE International Symposium on Distributed Objects and Applications*, Italy, September 2001.

[18] S. Oaks, *Java Security*, 2^{nd} *Edition*. Sebastopol, CA: O' Reilly and Assosiates, Inc., 2001, pp. 18–130.

[19] T. Sander and C. F. Tschudin, "Protecting mobile agents against malicious hosts," in *Mobile Agents and Security*. London, UK: Springer-Verlag, 1998, pp. 44–60.